Sami Näppä

Redefining the States of Test Places in LTE Receiver Continuous Integration

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

9 May 2015

Helsinki Metropolia University of Applied Sciences

Author	Sami Näppä
Title	Redefining the states of test places in LTE receiver continuous integration
Number of Pages	47 pages
Date	9 May 2015
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructors	Mikko Harila, Team Leader Jaana Holvikivi, Principal Lecturer

The need to perform automated testing on a software project has created the urge to develop automated testing environments. Often if these environments are developed within a software company they are very customized, and specifically created for that company's testing needs only.

The purpose of the practical work that was done in this final year project was to simplify the logic of the in-house programs that are used in test place management. These test places are used in continuous integration to test the development code. The implementation is made according to a pre-defined specification that introduces five new objects that hold the information that is used to determine the current state of a test place. According to the specification, one of the new objects will hold a value that represents the test place's availability for testing purposes from the continuous integration point of view. The value of the object that represents the continuous integration availability is set according to the combination of the values of the other four objects. The values of these four objects are updated according to the commands given by the user, or according to the changes that has happened in the system. As a result of the modifications done the overall logic complexity can be reduced from the programs used in continuous integration.

This thesis includes a theoretical part that explains briefly the history of mobile networking and the current trends of mobile communication. This thesis also includes theoretical information about the Agile software development practices, and how the Agile methods are different compared to the older and more traditional software development practices, such as the Waterfall model.

As a result of the work done, a modified test place reservation system was developed. The new specification has undergone a minimal modification in order to keep the stored information relevant from the system point of view.

Keywords	LTE, continuous integration, agile

Tekijä Otsikko Sivumäärä Aika	Sami Näppä Jatkuvassa integraatiossa käytettävien testipaikkojen tilojen yksinkertaistaminen 47 sivua 9.5.2015
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaajat	team leader Mikko Harila yliopettaja Jaana Holvikivi

Automatisoitujen testausympäristöjen kehitys on seurausta tarpeesta automatisoida ohjelmakoodin testaus ohjelmistoprojekteissa. Useimmiten nämä ympäristöt ovat yritysten tekemiä ja soveltuvat vain yrityksen sisäiseen käyttöön. Tämä insinöörityö on tehty Nokia Oyj:lle, jossa automatisoituja testausympäristöjä hyödynnetään uuden ohjelmakoodin testauksessa.

Insinöörityön teoriaosuudessa tarkastellaan matkapuhelinverkkojen historiaa sekä nykyisiä kehityssuuntia. Työn teoriaosuus myös käsittelee ketteriä ohjelmistokehitysmenetelmiä ja jatkuvaa integraatiota sekä tarkastelee, kuinka ketterät ohjelmistokehitysmenetelmät eroavat perinteisimmistä menetelmistä.

Insinöörityön tavoitteena oli muokata yrityksen sisäisessä käytössä olevien ohjelmien logiikkaa ennalta kehitetyn määritelmän mukaisesti. Näitä edellä mainittuja ohjelmia käytetään testauspaikkojen varauksenhallinnassa ja ylläpidossa. Nämä testauspaikat ovat osa jatkuvaa integraatiota ja testauspaikkoja käytetään ohjelmakoodin testaamisessa. Testauspaikkojen tietoja säilytetään tietokannassa ja nämä edellä mainitut ohjelmat muokkaavat, sekä näyttävät tietokannassa olevia tietoja. Edellä mainittu uusi määritelmä määrittelee viisi uutta tietokantakolumnia olemassa olevaan tietokantatauluun, joiden avulla vanhoista ja epäselkeistä kolumneista pääsisi eroon. Näiden uusien kolumnien avulla pystytään määrittelemään jokaisen yksittäisen testipaikan tila ja soveltuvuus testausta varten. Uuden määritelmän mukaisten muutosten seurauksena koko jatkuvan integraation logiikka, ohjelmien logiikka sekä tietokantataulun rakenne yksinkertaistuu.

Projektin aikana kehitettiin määritelmän mukainen varausjärjestelmä. Alkuperäinen määritelmä on läpikäynyt pienehköjä muutoksia, jotta tietokannassa säilytettäisiin vain kokonaisuuden kannalta oleellinen tieto. Insinöörityön kirjoitusprosessin aikana uutta järjestelmää ei otettu käyttöön.

Avainsanat	LTE, jatkuva integraatio, ketterä ohjelmistokehitys

Contents

1 Introduction

Since the dawn of the mankind, the humans have had the need to communicate with each other over long distances. Throughout the years the communication methods have evolved to modern wireless communication. For most people the usage of a mobile phone is an obvious part of everyday life. Yet not many people stop and think how a mobile phone actually works and, more importantly, how it connects you and your friend even though you are thousands of kilometers apart.

The possibility to use services provided via the Internet with a mobile device has created the urge to create and develop mobile networking standards enabling faster connections. The reason why the usage of mobile networks has been growing rapidly is because the design and usability of the mobile phones have reached a level that makes the usage of the Internet a pleasant experience. This has spawn out a considerable amount of competition between software companies that are now making applications that utilize the possibility of network usage. Telecommunication equipment manufacturing companies are trying to get ahead of the curve and provide devices capable of greater connection speeds than the competitors. While the coverage of the 4G network is not being even near the coverage of 3G, the newspapers and businesses are already talking about the future and 5G.

Telecommunication companies also have the need to execute comprehensive testing on the new base station modules before they can be deployed to the customer. One possibility for this is to establish a testing environment that automatically tests the new features on actual base station modules. This can be achieved by having a practice called continuous integration. The work done in the final year project described in this thesis modifies the logic of an existing continuous integration process and auxiliary programs connected to it.

The goal of this thesis is to modify the logic of the in-house programs that are used in test place management and to implement new columns to a database table that holds the information about every test place. These new database columns hold the values that are then used to determine the status of each test place. The idea behind this new implementation is to have a less complex continuous integration logic and less complex database table structure.

2 Introduction to LTE and mobile networking

The introduction of mobile telecommunication systems was in the early 1980s. The first generation (1G) systems used analogue communication techniques similar to those used in analogue radio. In the first generation mobile networks the individual cells were large and they did not use the available radio spectrum efficiently so the capacity was low, at least by considering today's standards. The user devices such as mobile phones were large and expensive and were marketed almost exclusively at business users. A more customer friendly direction was taken when the second generation (2G) of mobile telecommunications was introduced in the early 1990s. The second generation systems were the first to use digital technology which resulted in more efficient use of the radio spectrum. At the same time the customer devices became a lot smaller and cheaper. Originally these mobile phones were designed just for voice but were later enhanced to support the Short Message Service (SMS). The most popular second generation system was the Global System for Mobile Communications (GSM) which became popular throughout the world. Another notable second generation system was known as IS-95 (or cdmaOne) which became the dominant second generation system in the USA markets. [1.]

Success of the second generation systems happened at about the same time as the initial growth of the Internet usage. Network operators tried to bring these two concepts together which resulted in 2.5G. This 2.5G system that was built on the original ideas of the second generation introduced the core network's packet switched domain by modifying the air interface so that it could handle data as well as voice. The General Packet Radio Service (GPRS) incorporated these techniques into GSM. At the same time IS-95 developed into IS-95B. Also at the same time the Internet connection data rates available were progressively increasing. [1.]

The third generation (3G) was introduced in 2000. Third generation systems use different techniques for radio transmission and reception compared to the second generation. This resulted in increased peak data rates and more efficient usage of the available radio spectrum. The third generation systems became more popular after 3.5G systems were introduced in 2005. In 3.5G the air interface includes extra optimization which increases the average rate at which the user can upload or download information. [1.]

2.1 Mobile networks

The official name for the mobile phone network is public land mobile network (PLMN). This network is run by a network operator. The network architecture of Universal Mobile Telecommunications System (UMTS) and GSM is similar. Three main components of network architecture which are a core network, a radio access network and a mobile phone. The core network contains two domains which are the circuit switched domain that transports phone calls across the geographical region that the network operator is covering in a similar way to traditional fixed-line communication systems. The circuit switched domain communicates with the public switched telephone network (PSTN) so that users with a mobile phone can make calls to land lines and with the circuit switched domains of another network operator. Another domain of the core network is the packet switched (PS) domain. It is used to transport data streams, for example web pages or emails, between the user and external packet data networks (PDN) such as the Internet. The method of information transport between these two domains differs a lot. The circuit switched domain uses a technique called circuit switching. This technique sets a dedicated two-way connection for each individual phone call. The benefit of this is that the data rate is constant and delay minimal. However, it is rather inefficient even though it can handle scenarios where both users are speaking at the same time but it is usually over-dimensioned. It is inappropriate for data transfers in which the data rate may vary a lot. The packet switched domain uses a technique called packet switching. In packet switching, a data stream is divided into packets, each of these packets containing address of the required destination device. Routers use this address information in a network to forward the packet to the corresponding destination. The network's resources are shared amongst the users. This results in delays which occur when multiple devices try to transmit data at the same time. [1.]

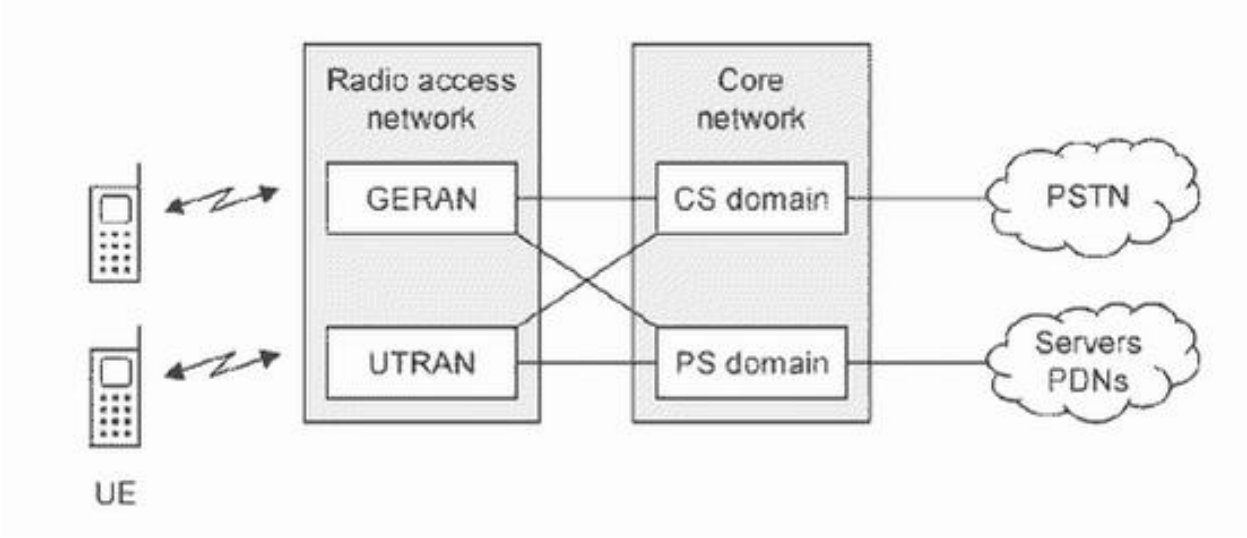

Figure 1. High level graph of UMTS and GSM architecture. Reprinted from [1.]

Radio communications with the user equipment (UE) and core network are handled by radio access network. In figure 1 there are two different radio access networks: GSM Edge radio access network (GERAN) and UMTS terrestrial radio access network (UTRAN). These radio access networks use different radio communication techniques but share a common core network between them. The communication interface between radio access network and user's mobile phone is called air interface or radio interface. The direction from network to mobile is called downlink (DL) and from mobile to network uplink (UL). [1.]

2.2 Radio access network

The most important part of the radio access network in UMTS is the base station, which is also known as Node B. A mobile phone network might contain as many as several thousand base stations altogether. Each of these base stations have one or more sets of antennas. These antennas are used in communication with the users' mobile equipment. A typical base station uses three sets of antennas to control three cells. Each one of these cells span an arc of 120°. Every cell has a particular maximum size determined by the maximum range at which the receiver can successfully communicate with the transmitter. A cell also has a maximum combined data rate which leads to the need of a different type of cells. Macrocells provide wide-area coverage in rural areas or suburbs where the need for high data rates or a high amount of users is not needed. Macrocells are usually a few kilometers in size. Microcells provide higher capacity for data rates and

they support a higher amount of concurrent users. Microcells are more suitable for dense areas and are only a few hundred meters in size. Picocells are used in large indoor environments such as offices or shopping centres and are smaller in size than microcells, usually less than 100 meters across. Femtocells are established by an end user purchasing hardware known as the home base station which can be installed to living quarters. Femtocells are only a few meters in size. [1.]

Transmission of data between a mobile phone and base station is done by using a certain radio frequency called carrier frequency. Carrier frequency occupies a certain amount of frequency spectrum known as the bandwidth. This means that if a mobile phone transmits data with a carrier frequency of 1960 MHz and a bandwidth of 10 MHz it would occupy a frequency range from 1955 to 1965 MHz. The air interface has to separate the transmission of the base station and the mobile phone in order to make sure that they do not interfere. In UMTS this can be achieved by using frequency division duplex (FDD) in which the base stations transmit on one carrier frequency and the mobile phone on another. Another option is to use time division duplexing (TDD) where the base stations and mobiles transmit on the same carrier frequency, but at different times. [1.]

When a mobile phone moves from one cell to another cell a switch must occur. This means that the mobile phone has to stop communicating with the old cell and start communicating with the new cell. Depending on circumstances, this can be achieved by using two different techniques known as handover and cell reselection. In UMTS it is possible for a mobile phone to communicate with multiple cells at same time. This state is known as a soft handover.

The base stations are grouped together by radio network controllers. Radio network controllers have two main tasks. One of their tasks is to pass the user's voice information and data packets between the base stations and core networks. The radio network controllers also control a mobile's radio communications by signalling messages that are invisible to the user, for example by telling a mobile to perform a handover from one cell to another. A network may contain tens of radio network controllers which control hundreds of base stations. In GSM the radio access network is similar in design. One difference is that the base station is called base transceiver station (BTS) and the controller is called base station controller (BSC). If a mobile phone supports both UMTS and GSM the network may hand it over between two radio access networks. This process is called

inter-system handover and can be invaluable in case the mobile phone is outside the coverage area of UMTS but is inside the coverage of GSM. [1.]

2.3 Introduction to LTE

LTE was originally designed by a collaboration of national and regional telecommunications standards bodies known as the Third Generation Partnership Project or 3GPP. The full name of LTE standard is 3GPP Long Term Evolution. LTE has evolved from an earlier 3GPP system known as the Universal Mobile Telecommunication System (UMTS) which is a third generation mobile cellular system (3G). UMTS itself was evolved from Global System for Mobile Communications (GSM). [1.]

The usage of mobile data has been steadily increasing during the past years. Figure 2 illustrates measurements made by Ericsson about the total mobile data traffic being handled by mobile communications networks worldwide. The figure shows data in petabytes (1 petabyte = 1000 terabyte = 1 million gigabyte) per month from January 2007 to July 2011. [1.]

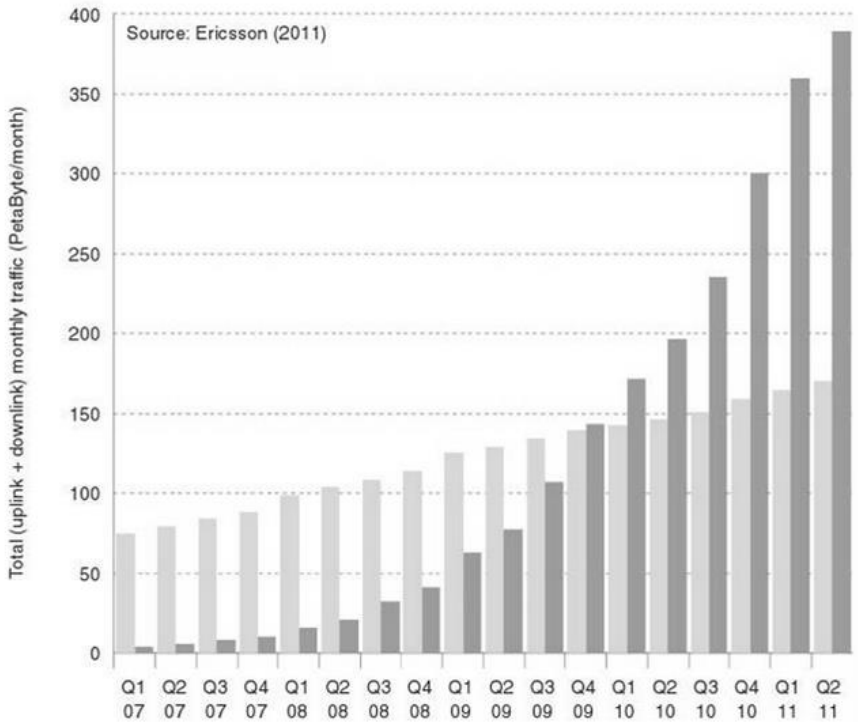

Figure 2. Data usage between January 2007 and July 2011. Reprinted from [1; 2.]

This upward trend visible in figure 2 is predicted to be heading towards a similar direction in the future as seen in figure 3 which shows forecasts about the growth of mobile traffic from 2011 to 2016.

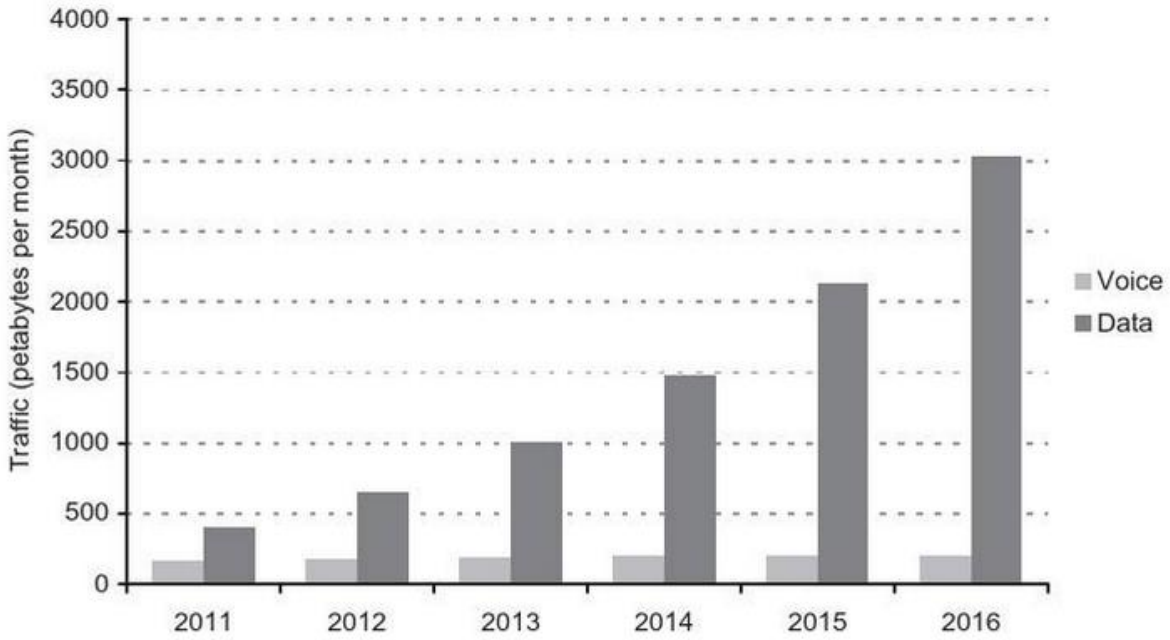

Figure 3. Analysts' predictions about the growth of mobile data usage from 2011 to 2016. Reprinted from [1.]

A big part of this increased data usage resulted from the fact that mobile phone manufacturers started to bring full touch screen displays to the markets. These mobile phones enabled more smooth usage of the Internet since the displays were a lot larger than they previously were. Also at the same time anyone was able to create their own applications or "apps" and share them worldwide via application marketplaces, such as App Store in case of Apple or Google Play in case of Android. A big part of these applications was also using data connections for sending messages between users in a simple messaging application or saving a high score in a game. The combination of high usage of data connections and the high amount of mobile phones resulted in data usage rising drastically.

In 1948, Claude Shannon discovered a theoretical limit of data rate in any communications system. This speed can be calculated by using equation 1.

$$C = B * \log 2\,(1 + SINR) \qquad (1)$$

Where C is the channel capacity in bits s^{-1}, B is the bandwidth of the communication system in Hertz and SINR (signal to interference plus noise ratio) is the power at the receiver due to a required signal, divided by the power due to noise and interference. In mobile communications C is the maximum speed a cell can handle and equals the combined data rate of all mobile phones in it. [1.]

There are three main ways to increase cell capacity in mobile networks. The first one is to establish smaller cells. This way the channel capacity per cell decreases and possible data transmit delays within the cell decrease. The second way is to increase bandwidth. However, the radio spectrum is managed by the International Telecommunication Unit (ITU) and by regional and national regulations. Due to the fact that there is only a finite amount of radio spectrum available and free frequencies are sparse this is not really an option. The third option is to improve the communications technique that is being used. The need for faster connections has been the driving force for new technologies from 1G to 3G and from 3G to LTE. [1.]

One of the benefits from switching 2G or 3G network to LTE is that there is no need to maintain two core networks for the circuit switched domain for voice and the packet switched domain for data. Even though it is possible to transport voice calls over packet switched networks using voice over IP (VoIP) technique, the network operator has the possibility to move everything to the packet switched domain, and reduce capital and operational expenses. [1.]

3 Continuous integration

Continuous integration (CI) is a software development practice where members of a team integrate their work daily to the mainline of the project. Each of these integrations made by a developer will have to pass automated software build and tests. Usually these integrations are committed to a version control system such as Git or Subversion. In case of errors the most important task is to fix them so that the automated build and tests pass. It is not recommended to have a failed integration build in continuous integration environment for too long. [3.]

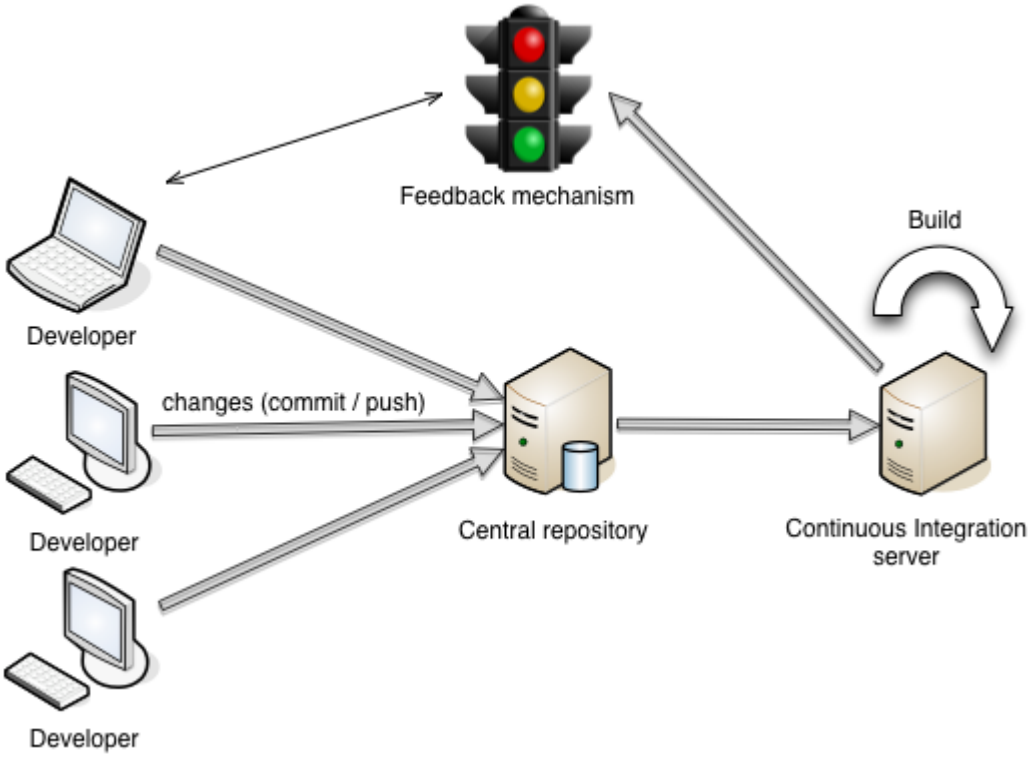

Figure 4. Continuous integration workflow. Reprinted from [4.]

A simplified diagram of continuous integration's workflow is presented in figure 4 above. Continuous integration was originally designed for the eXtreme Programming (XP) methodology, which is one of the Agile software development methodologies. [3.]

3.1 Agile software development

Agile is a collective term for methodologies and practices in software development that have come up over the past two decades. During the 1990s, when Waterfall project management methodologies were widely in use, the information technology industry was troubled by the high failure rate of software development projects and significantly exceeded budgets. The key factors for all of these software projects were over-planning, insufficient communication and so called all-at-once delivery. Traditionally the software product started with an extensive upfront documentation, which tried to take into account every possible aspect of the project in order to ensure that the developed software would match the customer's requirements. The upfront documentation usually took several months to produce and approve. The problem with having this kind of complete upfront documentation is that there is a tendency for the stakeholders to not clearly express their requirements which were then left for the software development team to assume and guess. Also one significant tendency for these projects was that the stakeholders wanted every possible feature that a software project could possibly have resulting in an impossible objective for the software development team to complete. Most of the traditional software development projects also suffered from the lack of communication between the business staff and developers. This led to a situation in which the developers were handed this massive upfront documentation and a deadline and they were then expected to produce complete and working software ready for the customer to test. This caused inevitable problems such as having possibly outdated software since the project may have lasted for months or even years. Also the business for which the software was initially created may have already shifted making the final software futile. [5.]

The Waterfall model was initially created with a goal to reduce business risk and failure rate in project delivery by requiring each step to be completed to management's satisfaction before proceeding on to the next step. In reality the use of Waterfall in a software development project increased the risk of failure by having an extensive upfront documentation, discouraging responsiveness and reducing responsiveness across the development team. Also the method of how and at which point the software was delivered to the customer increased the riskiness of the Waterfall model. Delaying the delivery to the very end of the project caused the possible costs of making changes to the software to be extremely high. Instead of validating the requirements throughout the project the Waterfall model forced the development team to use it to complete the project with an "all-

or-nothing" mentality. Figure 6 below is a graphical representation of the Waterfall software development model. [5.]

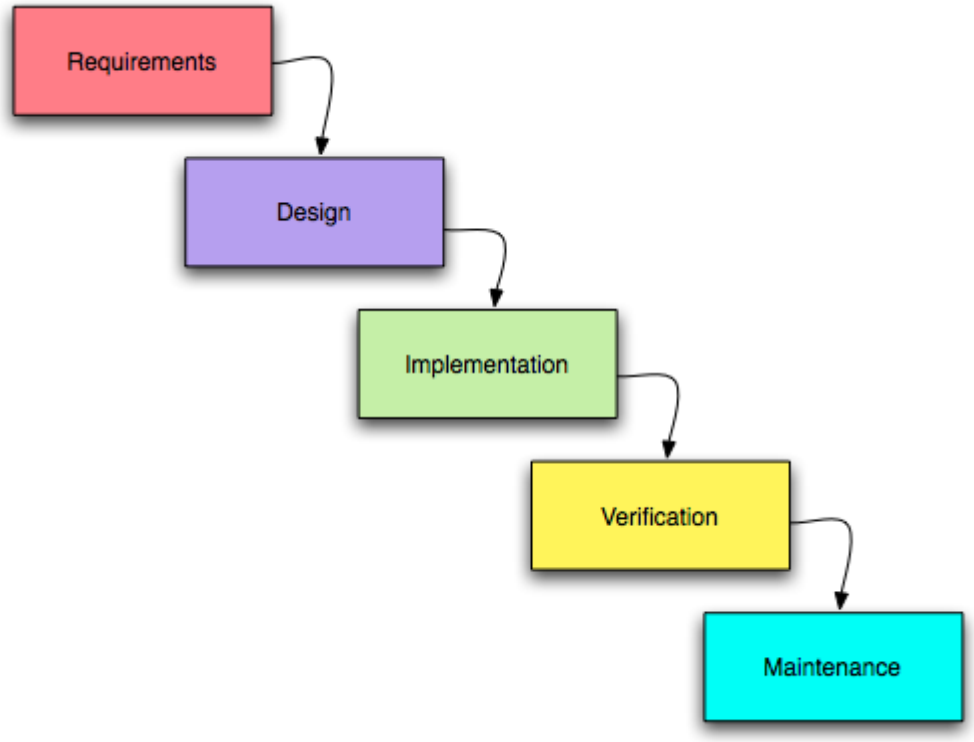

Figure 5. Graphical representation of the Waterfall model. Reprinted from [6.]

Comparing Agile software development techniques to more traditional software development techniques, such as the Waterfall software development model, Agile is designed to increase the quality, flexibility and the business value of software solutions. The Agile software development techniques are designed to address the problems that have previously bothered software development and service delivery activities. These problems are for example budget overruns, missed deadlines, low quality outputs and dissatisfied users. Even though the term Agile consists of many different methodologies and techniques all of them share the same basic objectives that are:

- To replace upfront planning with incremental planning
- To build in quality upfront
- To find out technical risks as early in the process as possible
- To minimize the impact of changing requirements
- To deliver frequent and continuous business value to the organization

- To entrust and empower staff
- To encourage and advance ongoing communication between the business areas and project team members [5.]

At a strategic level the Agile method has several benefits such as ongoing risk management by regularly confirming and adjusting requirements with the customer and ongoing control of budget expenditure by providing decision makers with the opportunity to review the completed feature after each iteration. Agile software development methodologies have been taken widely in use in thousands of companies mostly in the United States and Europe. Most notable technology companies that utilize Agile software development methodologies are Nokia, Yahoo, Google and Microsoft. [5.]

Some of the most common Agile methodologies such as Scrum, the Dynamic Systems Development Method (DSDM), Feature Driven Development (FDD), the Agile Unified Process (AUP) and Lean development include iterative strategies for managing the software development process. The most common of these methodologies is Scrum. Scrum is mostly used in the software development projects, but is also suitable for any project-based work. The workflow of the Scrum can be seen in figure 6 below. [5.]

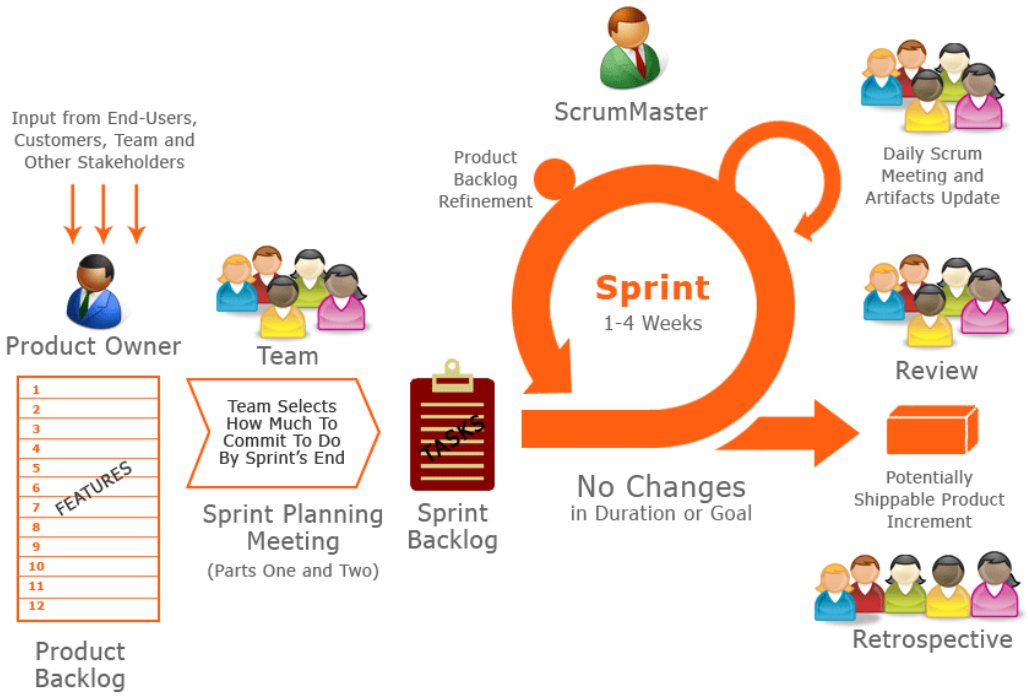

Figure 6. Scrum workflow. Reprinted from [7.]

The product owner, usually the customer, defines the wanted features of the software that the team will then be working on. The possible splitting of these features into smaller tasks is done by the team in product backlog grooming meetings. This splitting can be done with the idea that each of the new smaller tasks will be possible to complete within one sprint. The allocation of tasks to team members and the decision of what tasks are then taken into the sprint will be done in a sprint planning meeting. Team members will then have a predefined time period to complete the given tasks. This is called the sprint and usually sprint lasts from 1 to 4 weeks. Every day the team will meet in a daily scrum meeting and update their status on their current tasks and answer to the following questions: “what have I done since the last daily scrum meeting”, “what am I going to do before the next daily scrum meeting” and “are there any problems interfering the progression of my current tasks”. After the predefined time period of available working time is over the tasks that each member was doing should be completed. The team will then have a review and a retrospective meeting of the sprint. In these two, usually combined, meetings the members of the team may address possible grievance about the iterations or the current way of working. After this the sprint will start again with a new sprint planning meeting.

The sprint, which is the core of the Scrum, consists of sprint planning, at the very beginning of the sprint, the iteration in which team members have from 1 to 4 weeks to complete tasks given to them, the daily scrum meetings and the sprint ending meeting, which consists of sprint review and sprint retrospective.

3.2 Contributing in continuous integration

Contributing to continuous integration is done by copying the “mainline” of the project to developer's machine. This mainline is a term referring to the latest version that can be found in the version control system. After copying the mainline to the developer's own machine, the developer can then start making modifications to the code and adding or altering the automated tests in order to complete the given task. Whenever the developer is ready with the modifications the changes will then be built and tested on a development machine. Once the build and tests are passed successfully the changes can then be committed to the mainline. At this point the continuous integration can also have some pitfalls. The mainline code may contain new changes that were implemented after the developer copied it to the development machine. This means that the developer now has an outdated code with a new feature or implementation. Now the developer has to copy

the code from the mainline again using merging features provided by the version control system. After merging the new mainline to the changes the code has to be tested again and possible conflicts have to be fixed. After the developer has successfully synchronized the changes with the mainline, a commit can then be made. Unlike in normal software development, in continuous integration this is not the finishing point of the developer's work. After the changes have been committed to the mainline, the software will be built and tested again, but this time on an integration machine based on the mainline code. After this build and testing passes the work can be considered done. Any errors or clashes that have happened during this final building and testing should be quickly fixed with high priority since it is not beneficial to have a failed integration build stay failed for a long time. The idea in continuous integration is to have a stable piece of software available at all times that is tested and works properly and contains fewer software bugs. [3.]

3.3 Practices of continuous integration

3.3.1 Maintenance of a single source repository

Every software product that uses continuous integration should use a version control system. This takes off the effort of keeping the code up to date compared to for example having the code in a shared network drive. This means that all the files used in the project should be located in the repository. The basic rule of thumb in this practice is that it should be possible to build the project on a brand new machine by only fetching the code from the repository and not doing too many changes to the environment.

A version control system also allows its users to create branches in order to handle the different states and streams of the development. In continuous integration the rule of thumb is that the branch count should be kept as low as possible. Most of the time having only a single branch is enough. From this main branch the developers should fetch the project and to this branch the developers commit their newly made features. Noteworthy bug fixes are a rational reason for a project using continuous integration to have a new branch for.

3.3.2 Automation and testing of builds

In order to save time the build should be as automated as possible. Fortunately, many platforms have support for automated software building such as Make for Unix, MSBuild for .NET or Ant for Java [3.]. It is a common mistake in continuous integration not to include everything in the automated software build. The repository that contains the project should contain the configuration needed for the software build. A good building system analyses the code and determines what needs to be changed.

Even though the project builds successfully and does not give visible errors it does not mean that it will work as intended. Without any logic testing the software may contain logic flaws that may cause unexpected behaviour or memory leaks during the execution. This is the reason why the build should contain testing of the code logic. The software build should fail if testing detects any errors. During the last decade a software development process technique called test-driven development (TDD) has become more and more popular. This development technique changes the approach of how the programming in a project is done. In test-driven development the test cases of every function are written first. These so called unit tests are used to test individual units of the source code. The task for unit tests is to test the logic of the functions that are written to the production code later. This means that in test-driven development the production code is being written to pass these tests. When the unit tests are run the following output can be seen. This output indicates that the unit tests were successful. [8.]

```
[saminap@edunix pyunit_unittest]$ python calculatorTest.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Figure 7. The output of a successful unit test. Screenshot [9.]

Unit tests will fail when some part of the code that is being tested contains a logic flaw. Whenever this happens the unit tests will print error information and explain where the flaw can possibly be found.

```
[saminap@edunix pyunit_unittest]$ python calculatorTest.py
F
======================================================================
FAIL: testSum (__main__.TestCase)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "calculatorTest.py", line 11, in testSum
    assert self.calc.sum(6,10) == 15
AssertionError

----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (failures=1)
```

Figure 8. Output of the failed unit testing. Screenshot [9.]

Many programming languages nowadays provide a unit-testing framework for code testing. For example Java has multiple unit-testing frameworks available such as SpryTest, Jtest, Junit and TestNG. [8.]

3.3.3 Execution of building and testing after commits

By increasing the amount of commits to the mainline the amount of testing also increases. This means that the team gets more frequent tested builds and the mainline stays in as healthy state as possible. The developers should also time their commits so that they are still present in case of failure during testing or building. The developer who made the commit is responsible for fixing the possible errors that may have occurred. In continuous integration this can be ensured by using a manual build or having a continuous integration server. In case of a manual build a developer who committed the new feature goes to the integration machine and ignites the building and testing of the newest version of the project. Only after the build and tests have passed can the developer be sure that the new feature was stable and free from software bugs. In case of having a continuous integration server this previously mentioned procedure will be automated. Every time a developer makes a commit, the building and testing will be started automatically. Once the building and testing is ready or an error is found the system notifies the developer about the outcome and the possible follow-up actions will be started if needed. [3.]

As a result of the project growth the building and testing times increase. In continuous integration the aim is to have a system that provides rapid feedback. If building and testing of the project takes several hours some precautions should be made. Many times a bottleneck in testing is the usage of a database. This can be avoided by having a deployment pipeline that enables multiple builds to be done in sequence. A commit to a repository triggers the first build. This build will be executed without profound testing and will not be able to detect all the bugs that may be present. The main goal for this kind of approach is to have a balance between finding the bugs and having a fast build so that other developers may start working with the newly built and, in some cases, only marginally tested source code. The first stage of this kind of testing could contain only compilation of the source code and run localized unit tests without database access in order to achieve a timeline of less than 10 minutes. The second part of testing builds the source code and uses the database in order to involve more end-to-end like behaviour. This second part of testing has a larger timeline compared to the first case. If the first part of testing was successful but bugs or errors are found from the second part then the team aims to fix found bugs as fast as possible while keeping the first part running. [3.]

3.3.4 Transparent system

Communication is a vital part of continuous integration. By providing some sort of knowledge for every team member of the current state of the project the error handling and bug fixing become easier. This can be achieved by having an extensive logging of the builds, unit testing, integration testing and the whole system in general that everyone can access. [3.]

In order for the developers to see what state the project is in it is a widely used technique to have a graphical user interface of some sort that contains the information about the latest builds and tests. Usually this graphical user interface can be for example a website that contains this information. This website may also show which of the unit tests or software component tests were faulty and who made the commit. By having a website the developers may access the status information with ease. Jenkins also provides a web interface that shows the statuses of the recent builds.

3.4 Jenkins

Jenkins, which can also be referred as Hudson which was the original name, is an application that automatically monitors the executions of building a software project or jobs run by Cron. Cron is a program for Unix-like operating systems that executes scripts or commands automatically according to the user's directive. Jenkins was created in order to improve the effectiveness of the continuous integration process. Jenkins provides functionality for building and testing software projects continuously and monitoring the execution of Cron jobs. Jenkins is written in Java and the whole Jenkins' functionality is included in one web application archive file or .war file. This means that no further installation besides Java Runtime Environment or JRE is needed on the server side. Jenkins provides a web graphical user interface which can be used to access Jenkins settings and which also contains a lot of information and manuals about Jenkins. [10; 11.]

Monitoring Jenkins jobs can be achieved by having an RSS feed or having the system to send an e-mail so in case of failure the users will be notified instantly. Jenkins also provides a plugin support for 3rd party plugins which means that users are able to write their own plugins to Jenkins.

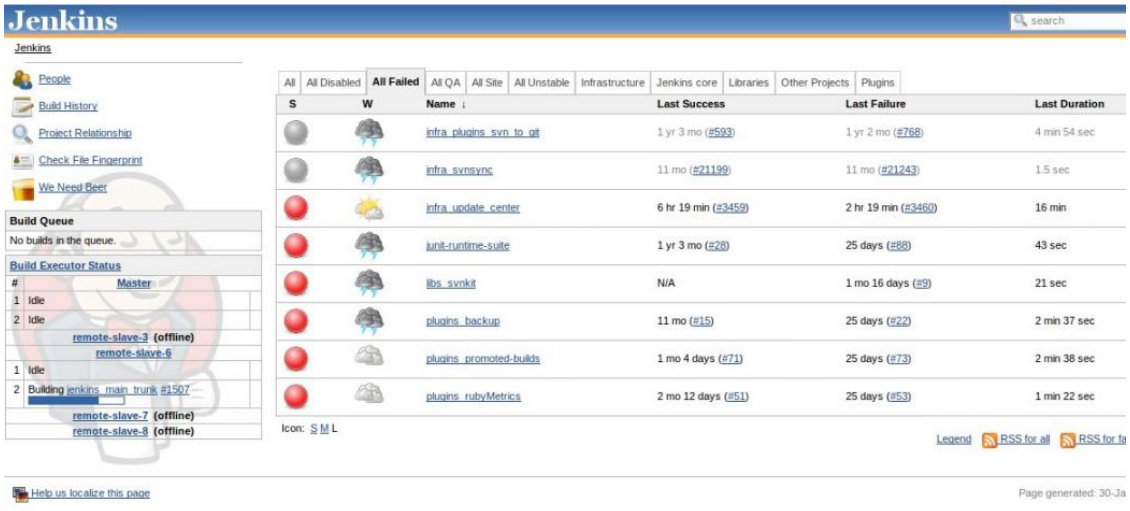

Figure 9. Jenkins web interface. Reprinted from [10.]

The Jenkins web interface shown above in figure 10 contains a list of current jobs. In this case the red or gray ball shown in the list represents the status of the last build. Red represents that a failed build has occurred and the gray represents that the build was possibly disabled or aborted. The icon seen on the right side of the status ball represents the stability of the latest Jenkins builds. The storming icon in this case indicates that the

last builds have not been very stable while less cloudier icons indicate that the recent builds were more stable. Users may customize the view of jobs by creating a tab to the jobs list. In a screenshot of Jenkins in figure 10 a tab named “All failed” is currently shown. Navigating in the Jenkins web interface can be done via the menu located in the upper left corner. All the jobs and the progress meters are listed under the menu if it is currently running a job. [10.]

3.5 Continuous integration in Nokia LTE base station receiver context

In the development of the Nokia LTE base station receiver, the continuous integration process is customized due to the restrictions of Jenkins. This process mainly consists of two parts: CIAPP responsible for the dynamical operation of the system (such as scheduling, running the builds and testing) and the user interface CIUI. The need for scheduling is the main reason for the existence of the CIAPP. CIAPP also handles tasks such as monitoring the version control system, in this case Subversion (or SVN), compiling the code, putting tests to a queue and executing tests. CIAPP is a vital part of this customized continuous integration process and it is run on Linux servers. [12.]

The test place computers contain a Java application that establishes a socket connection to Jenkins. The Jenkins main server can issue commands via this socket connection. Developers may, however, disable the continuous integration usage on a particular test place. Whenever the continuous integration usage is disabled on a test place the Java application that establishes the connection to Jenkins is closed. Whenever the connection to the Jenkins is allowed, or user’s reservation has ended, the Java application will be opened again and the connection to Jenkins servers will be established. CIAPP also contains a script called CiMonitor which is, despite having the same name, different than the CiMonitor that will be discussed in this thesis later. CIAPP’s CiMonitor determines which of the test place computers will be selected for testing. CiMonitor’s logic utilizes the information that is stored in the database. This database holds the information about the module type of the connected base station. CiMonitor uses this information while selecting the test place computer on which the tests will then be executed. CiMonitor contains a circular buffer that holds the information about which of the test places were triggered previously so that the distribution of the test runs between the test places is even to some extent.

The feedback mechanism displayed in the continuous integration workflow in figure 4 above is accomplished by having a web interface that contains the results of all the tests that have been executed. This interface is called CIUI and a screenshot of CIUI is presented in figure 10 below. Note that some of the information seen in figure 5 below is censored due to business privacy.

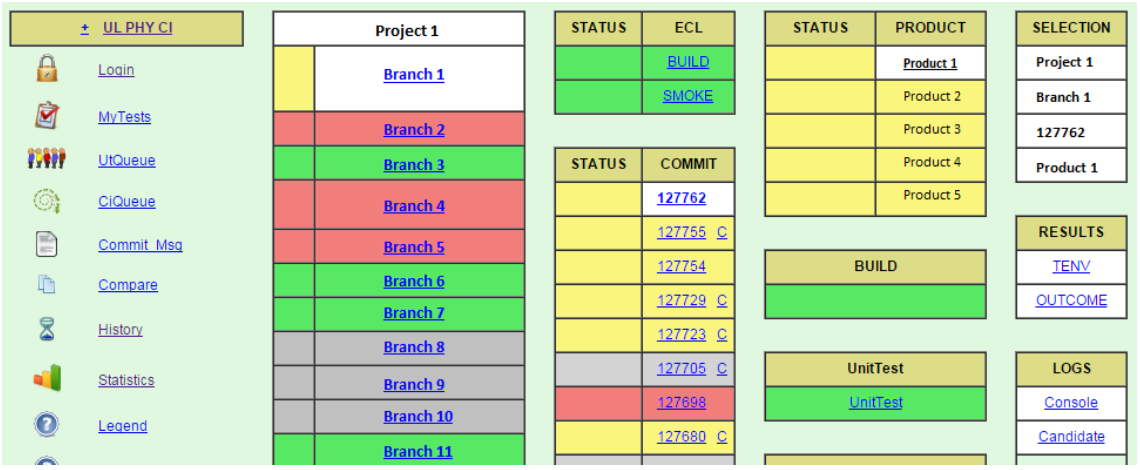

Figure 10. CIUI main view. Screenshot [13.]

The version control (SVN) branches that are currently used are listed in the first table from the left, under the heading “Project 1”. The background color of each of these branches represents the build and testing status of the latest commit. The red color represents that the latest commit in that particular branch has failed. The green color represents that the latest commit was successful while yellow or blue, depending on user settings, represents that the testing is ongoing or that there are queued tests on that particular branch. CIUI makes the continuous integration process transparent for all its users since it contains links to build and testing logs for every commit. Developers and CIUI users may configure their personal settings from CIUI. These settings contain a selection of which system emails the user wants to receive. For example the user can choose whether or not to receive emails about a failed build. The white background color seen in the branch, commit and product table indicates that the current branch, commit and product has been selected. This means that the log links will be for that particular commit and product.

4 Test place reservation system

The test place reservation system consists of four different in-house programs which are RemoteUser, RxUser, CiMonitor and Watchdog. These programs have different kinds of tasks which include reservation of a test place, enabling or disabling continuous integration usage, starting of a maintenance mode in the test place and monitoring test place reservation expiration. The communication between these programs is handled by sockets and by having a single database table that holds the information about every test place PC that is connected to the system. Most of the test places that are found in this database table are used in continuous integration. Usually these test places have a certain base station connected to them and the information about this connected base station is also stored in a database table. This allows the continuous integration system to direct the system component tests to a test place PC that has the correct base station type connected to it. Developers may utilize the performance of a test place and execute tests by temporarily disallowing the continuous integration usage from the test place by using the reservation system. Developers may also issue customized testing commands via an in-house program called Phyt. With Phyt the user can test the newly made modifications before committing them to the official software branches. Phyt forwards the tests to the continuous integration system that then handles the scheduling, queuing, the selection of a proper base station module type and finally the reporting. Due to the scheduling provided by the CIAPP the developers are able to discontinue the continuous integration usage on a test place even if the testing is ongoing. Whenever this happens the CIAPP handles the rescheduling of the discontinued test job. The test place reservation system consists of approximately 100 test places and most of them use the Linux operating system.

The goal of this final year project is to modify RemoteUser's, RxUser's, CiMonitor's and Watchdog's logic according to a pre-defined slave state specification. This specification was developed in order to reduce the complexity of the selection of which test places were available for continuous integration usage. This specification also defined new database columns to a database table. Once these changes are accomplished the old database columns may be removed, resulting in reduced complexity in the table's structure. This also allows the continuous integration system to have reduced complexity in its logic while determining which of the test places are online or offline and which of the test places are available for testing. The simplified version of the relationships between these

previously mentioned in-house programs used in the test place reservation system is presented in Figure 11 below.

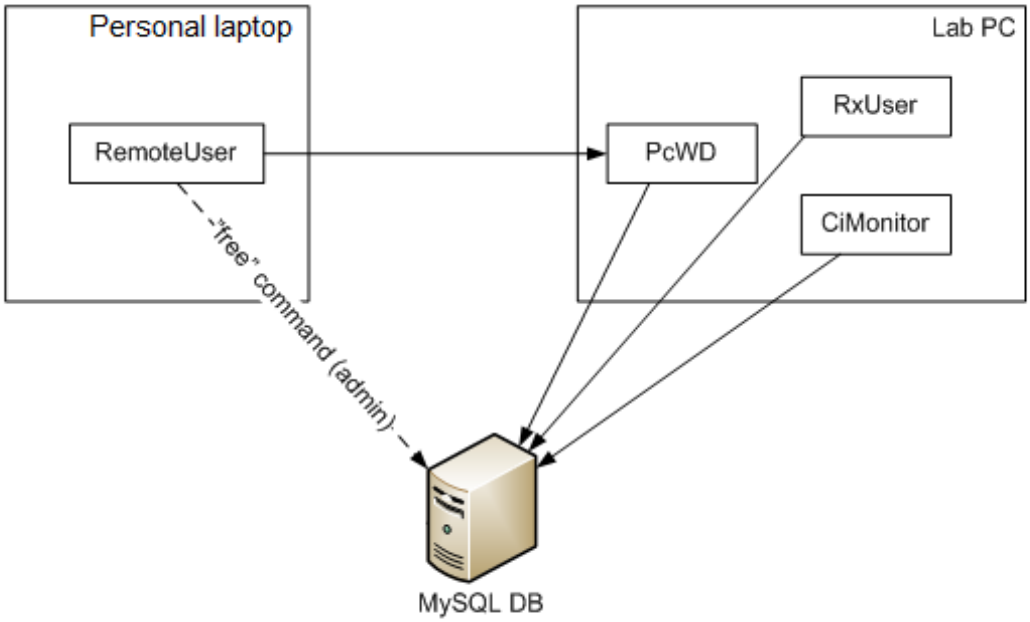

Figure 11. Test place reservation system. Reprinted from [14.]

The MySQL database presented in figure 11 above is used as inter-process communication (IPC). Some of the IPC features are also achieved by having socket connections. The MySQL database contains values which are used by various different programs to determine the state of that particular slave. Most of the test places have a CiMonitor which in case of Windows contains the functionality which will be executed when that particular test place is set to maintenance mode. Most of the functionality that will be executed when commands are given from RemoteUser is in Watchdog (PcWD in figure 11) since it will be running on every test place regardless of the operating system. Watchdog is run as a service on Windows and daemon on Linux. The connection which will be used to transmit commands between RemoteUser and Watchdog is implemented by using sockets.

The graphical user interface of RemoteUser, CiMonitor and RxUser is done by using PyQt, which is a Python binding for the Qt framework. The Qt toolkit is a cross-platform application and graphical user interface framework originally developed by Trolltech. Since Qt is C++ native, PyQt was developed to combine the advantages of Python and Qt. [15.]

4.1 RemoteUser

RemoteUser is a program that provides a graphical user interface. Remoteuser is used on a personal laptop and it is mainly used to make reservations for test places. RemoteUser is also the main tool for administrators in test place maintenance. The main view of RemoteUser consists of three tabs which are “Remote PCs”, “Remote BTS settings” and “Remote BTSs”. RemoteUser can be used to reserve a test place, edit the current reservation, set the test place to maintenance mode and allow or disallow CI usage. Users can also start a remote desktop connection from RemoteUser by clicking the start button in the “RDC” column. RemoteUser will then start a program called Remote Desktop Connection if the test place’s operating system is Windows and NoMachine in case of Linux.

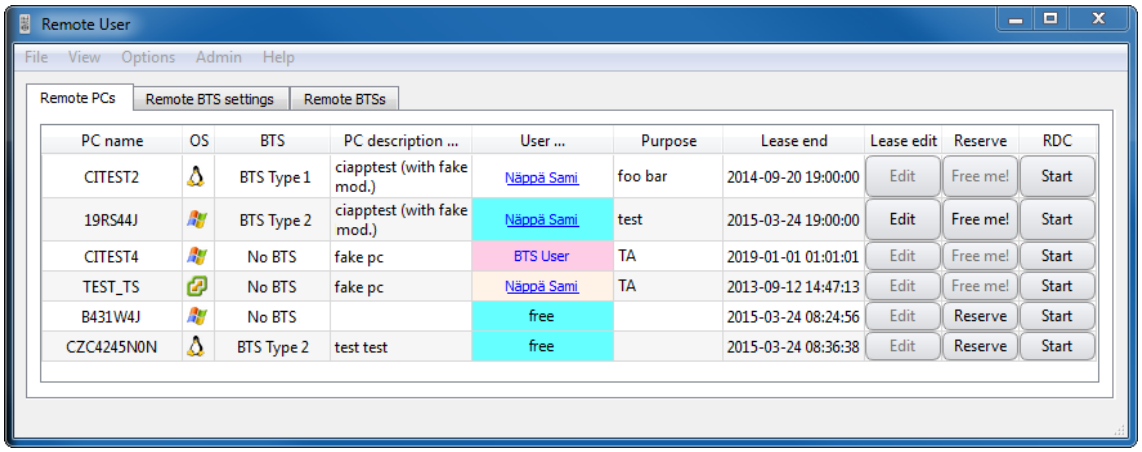

Figure 12. RemoteUser graphical user interface. Screenshot [16.]

The colors seen in the “User” column on the Remote PCs tab in figure 12 above can be used to determine which state the test place is currently in:

- Turquoise, this test place can be used by CI to run tests.
- Gradient turquoise, CI tests can be run on this test place after 20:00
- Pink, this test place is in maintenance mode.
- Light orange, this test place cannot be used to run CI tests but it can still be reserved.
- Transparent, the CI is not enabled in this test place.

RemoteUser runs a thread that updates the Remote PCs tab every second. During this update the information about every test place is fetched from the database so that the data displayed in RemoteUser will be as real time as possible. Every test place shown in this view also has a mouse right click menu which can be used for example to release the PC from continuous integration usage, change PC description or allow continuous integration usage if it has been previously disabled. Figure 13 below represents the functionality provided for the administrators by the mouse right click menu. If the user is not an administrator of the system, then the commands “Change user”, “Start maintenance” and “Start maintenance + BTS OFF” will not be visible and thus cannot be issued.

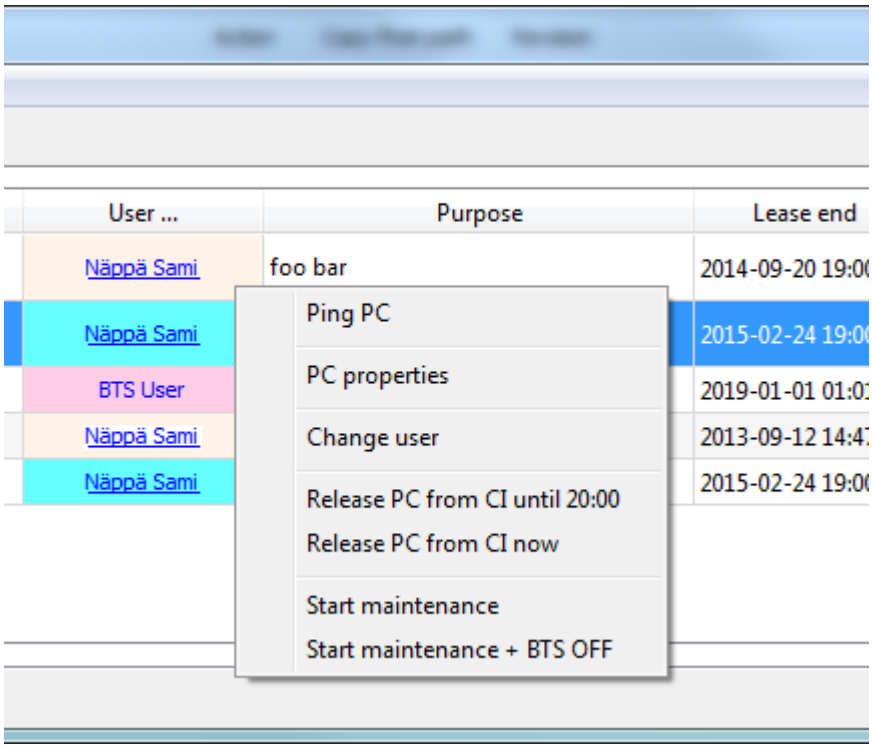

Figure 13. RemoteUser mouse right click menu. Screenshot [16.]

The database contains a table that contains usernames that are considered to be administrators. Administrators have more functionality given. For example, administrators are able to set a test place to the maintenance state when using the mouse right click menu in RemoteUser. In figure 13 above RemoteUser is run in debug mode and the user is set to be an administrator.

4.2 RxUser

RxUser is a graphical user interface run in most of the test places. RxUser has similar functionality as RemoteUser but it is located in the test place PC. The user can make or quit a reservation with RxUser, disable or allow continuous integration usage and update reservation informations. RxUser shows important information about a particular test place such as information about the base station that is currently connected to the test place. Figure 14 shows RxUser's main view. In this figure RxUser is run in the debug mode. The debug mode can be turned on by changing the debug variable to true in the RxUser code.

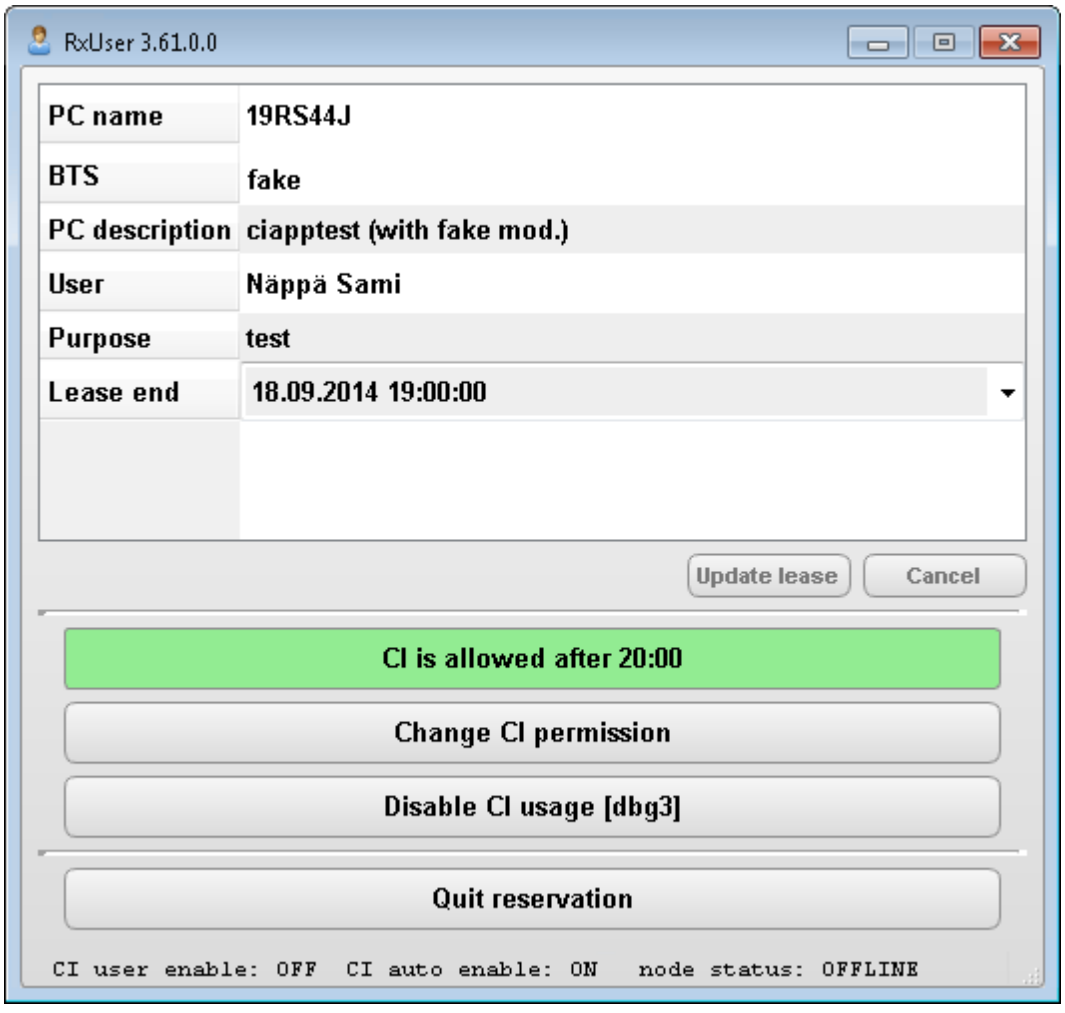

Figure 14. RxUser in debug mode. Screenshot [17.]

The debug mode displays more information in RxUser's graphical user interface and allows reservations to be made without giving the username and the password. Previously RxUser also handled the checking of the connected base station module type.

Whenever the information about the connected base station module type changed, RxUser modified the database accordingly. During this project the base station checking was removed from the RxUser and was added to the Watchdog. RxUser also logs the commands that were issued using the graphical user interface.

4.3 CiMonitor

CiMonitor is run on test places that are used to run continuous integration tests. CiMonitor handles the interaction between Jenkins and test place. CiMonitor handles functions such as:

- Starting and stopping the Jenkins slave process
- Calling the Jenkins monitoring job
- Updating the current state of Jenkins to database
- Automatically starting Jenkins slave process when the test place is free.
- Displaying top window and full screen window when necessary.
- Recording the uptime.

The figure 15 below illustrates a simplified flow chart of CiMonitor's main loop. This main loop will be run every 5 seconds.

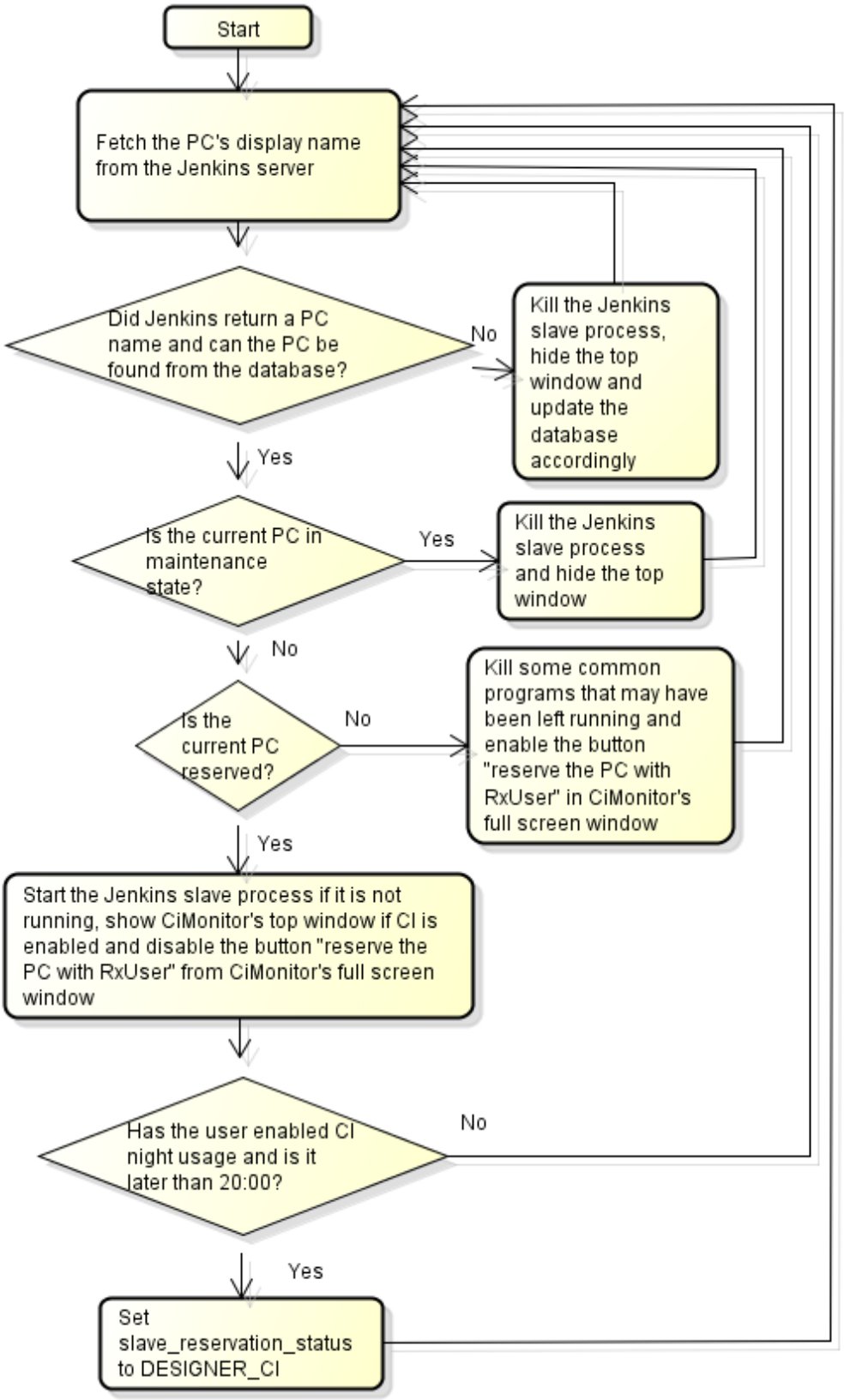

Figure 15. A flow chart illustrating the main loop of CiMonitor

CiMonitor also displays a full-screen window whenever the test place is not reserved. This full-screen window contains buttons for making a reservation or releasing the test place from Jenkins. CiMonitor's full screen window is displayed in figure 16. In this figure the test place is reserved and the button “Reserve the PC with RxUser” is disabled. If the test place is not reserved then the reserve button would be enabled and “Release the PC from CI” and “Hide this Window but let CI still use the PC” is disabled. By clicking any of these buttons the full screen window will be hidden, and if the user has made a reservation and continuous integration is enabled, then the small CiMonitor window will be shown. If the continuous integration is not enabled or the user has not made a reservation, the small window will not be shown. The full screen window will be automatically shown again in 30 seconds if the user has not made a reservation by then.

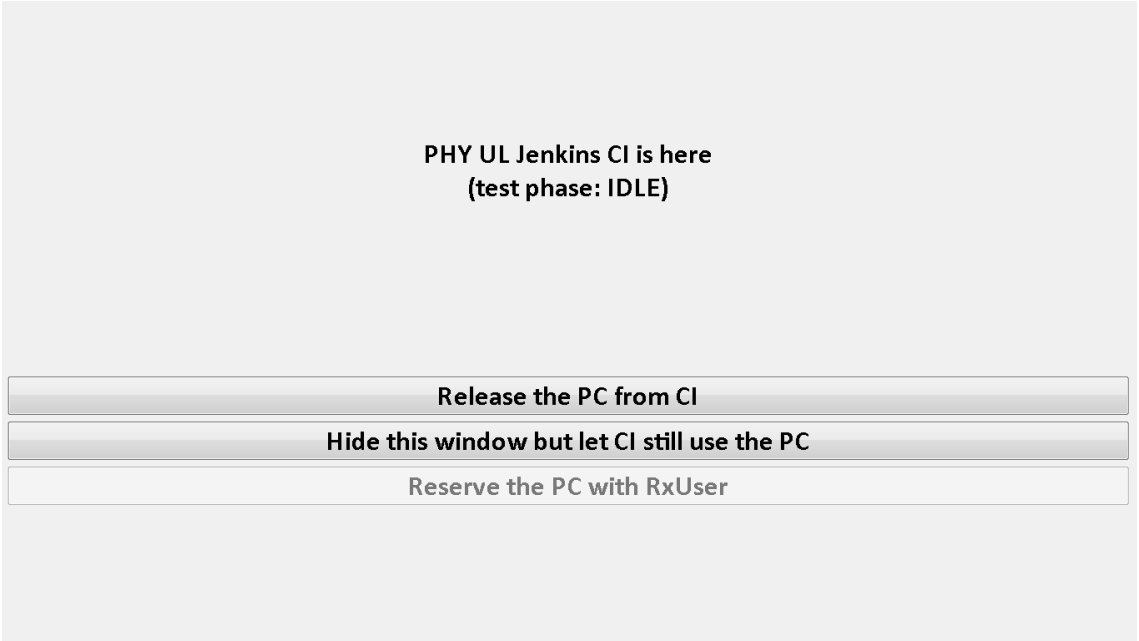

Figure 16. CiMonitor’s full screen window. Screenshot [18.]

While the test place is reserved, the CiMonitor calculates the total time that the user has allowed continuous integration to be run in that particular test place and updates it to the database. Previously this time was saved to the database as MySQL time type. The time type value that is provided by the MySQL can store values between ‘-838:59:59’ and ‘838:59:59’ and requires 3 bytes of storage. It was decided to change the implementation so that the calculated time was stored as the amount of minutes the test place has been in continuous integration usage during the reservation. MySQL type of unsigned medium integer was decided to be used in this case. A medium integer requires 3 bytes of storage and can store values from -8,388,608 to 8,388,607 as a signed medium integer and

values from 0 to 16,777,215 as an unsigned medium integer. [19; 20.] By storing the amount of minutes the test place has been in continuous integration usage it is possible to have a test place that has been in continuous integration usage for 16,777,215 minutes which is approximately 32 years. This can be considered a value big enough for this particular usage. By changing the logic of the calculation the need to create a function that would transform the given minute amount into a format of "x Days, HH:MM" arose. Listing 1 below illustrates the solution that was designed to handle this problem. This function returns a Python data type tuple that contains the days, hours and minutes that were calculated from the minute amount that was given as a parameter.

```
def toTime(minutes):
    d = minutes / 1440
    h = minutes % 1440 / 60
    m = minutes % 1440 % 60
    return d, h, m
```

Listing 1. A code example illustrating how to transform the given minute value to days, hours and minutes

By storing the continuous integration usage time as minutes in the database a new problem occurred. In order to save processing power, CiMonitor's main loop has the Python's `sleep(5)` function issued after every execution. This means that the execution of the main loop will always take at least 5 seconds since the `sleep()` function with a parameter 5 will halt the main loop for 5 seconds after the execution. Also the execution of the loop consumes a variable amount of time depending on how heavily the test place is currently used. Since it is not safe to assume that the loop will get executed at an even minute, more logic was needed to capture the so called "overflowing" seconds. Whenever 60 or more seconds has passed since the last update the database will get updated. The variable that calculates this difference is often more than 60 resulting in an incorrect value when having a reservation that lasts for several hours or even days. This is why a new variable that holds the remainder of the calculator variable was introduced. Whenever this new variable reaches the value of 60 or more the database will be adjusted accordingly. After this the variable that holds the overflowing seconds will be reset back to 0. This increases the accuracy of the timing calculations.

The time of how long the continuous integration has been enabled during a reservation is shown in RemoteUser by hovering the cursor over the test place. A screenshot of this

can be seen in figure 17 below. This functionality is achieved by using the function introduced in listing 1.

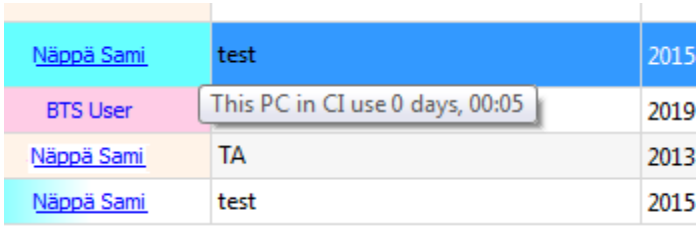

Figure 17. Hover display that indicates how long the test place has been in CI usage during the reservation. Screenshot [16.]

When the full screen window is not visible a smaller window will be shown. The smaller window displays information about the current test status. It also contains a button which can be used to open the full screen window. CiMonitor's small window is displayed in figure 18 below.

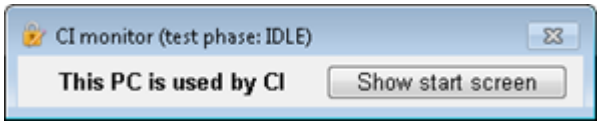

Figure 18. CiMonitor's small window. Screenshot [18.]

The relation between the full screen and the small window is that the small window should be shown whenever the user has a reservation and the full screen window is hidden. Once the user ends the reservation or the reservation expires then the full screen window will become visible and the small window will be hiddens.

4.4 Watchdog

Watchdog is run on every test place as a service in case of Windows and as a daemon in case of Linux and is running even though the user has not logged in. Watchdog creates a socket connection in order to receive remote commands from RemoteUser. Watchdog sends email informing the user once the reservation has expired and updates the database accordingly. Watchdog also queries the status of the current base station connected to it.

Figure 19 below illustrates the main loop of the new Watchdog. The added functionality compared to the previous Watchdog is that the Jenkins connection status will be determined by the Watchdog and then it will be stored in the database. The base station connection polling will be done by Watchdog instead of RxUser and the slave_CI_status will be set according to the slave state combinations that are mentioned later in this thesis.

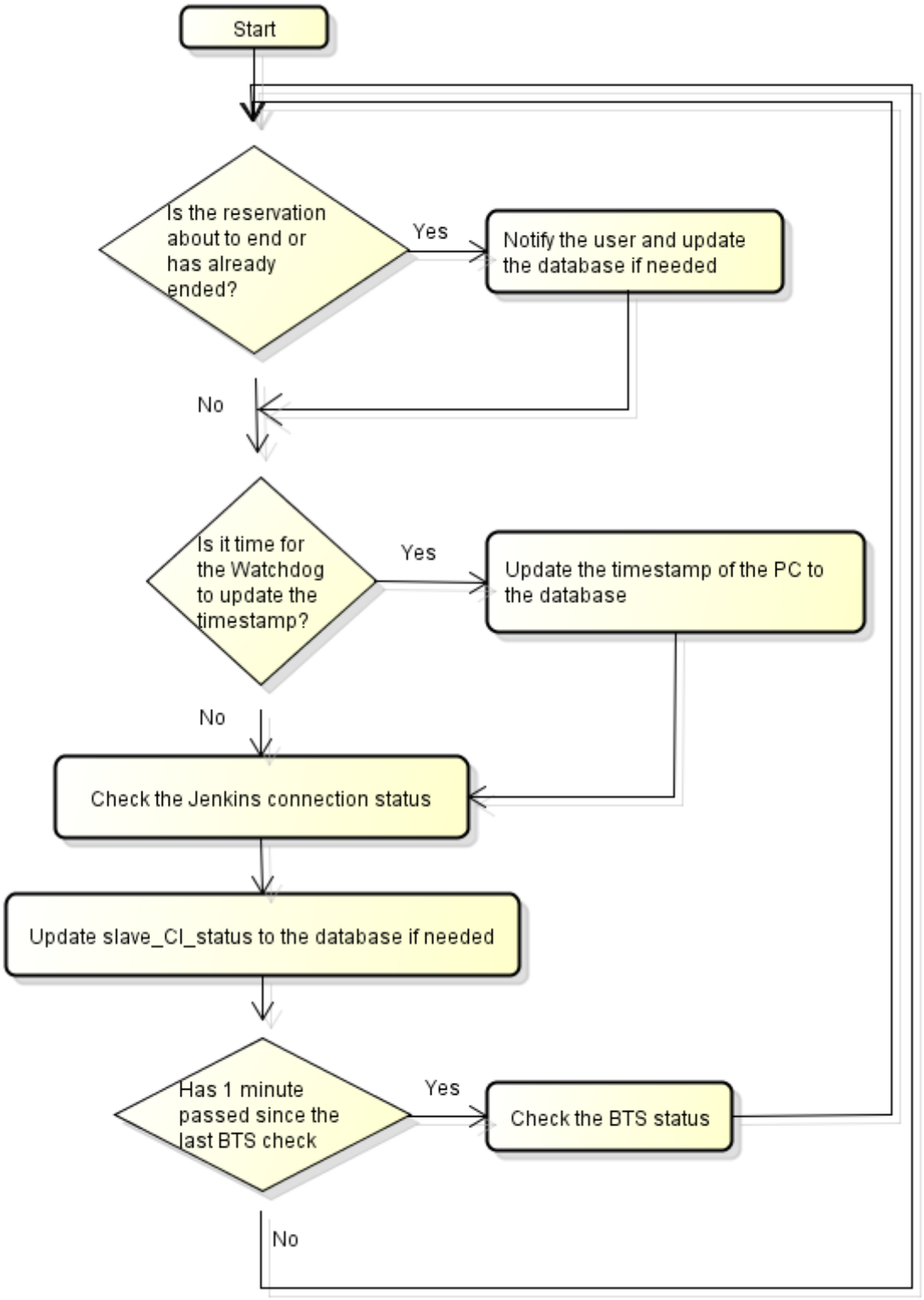

Figure 19. Flow char representing Watchdog's main loop

Watchdog will update the database value slave_CI_status if the combination of slave_reservation_status, slave_BTS_status and slave_jenkins_status indicates that the slave_CI_status should be ONLINE or OFFLINE and the current value is the opposite.

This loop will be run every 15 seconds and the base station polling every minute. By updating the current timestamp to the database during the Watchdog main loop the test place signals that the current node is up and running without problems.

During the check of user reservation status the reservation end time will also be checked. If the reservation ends in less than 15 minutes the user will be notified by an email that informs the user about the upcoming end of the reservation. Figure 20 below is a screenshot of an email sent by Watchdog that informs that the user's reservation is ending in 15 minutes.

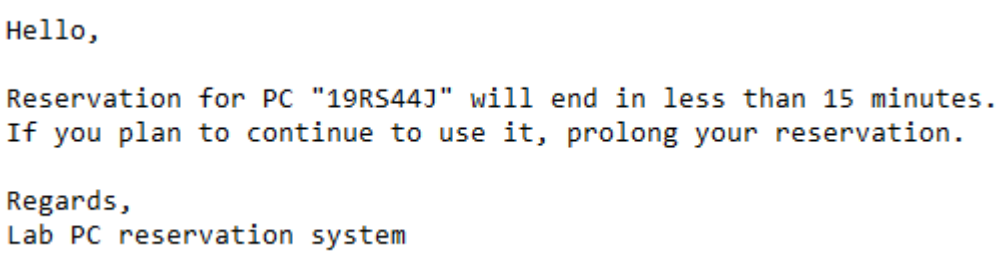

Figure 20. An email that indicates that reservation is about to end. Screenshot [21.]

Also when the user's reservation time ends, the user will be notified by another email. Figure 21 below is a screenshot of an email sent by Watchdog that informs that the user's reservation has ended.

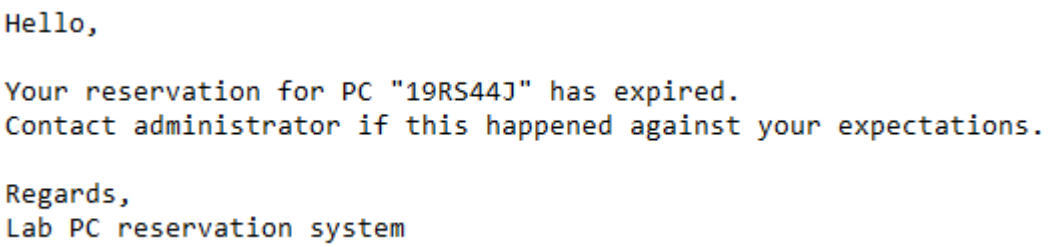

Figure 21. An email that indicates that the reservation has ended. Screenshot [21.]

Watchdog provides a functionality that checks if any base station is connected to that particular test place. If a base station is connected to a test place Watchdog will check whether or not the information about the base station is already present in the database. If the base station that was found is different than the current base station then the old base station information will be removed from the database and the new one will be

added. If the database does not contain any information about the connected base station, then the database will be updated. This base station status will be checked every minute.

4.5 Remote usage

Once the test place is reserved the user may utilize its performance via remote desktop access. Remote desktop access can be achieved by using Microsoft's Remote Desktop Connection (RDC) or NoMachine. The test places can also be accessed via remote access programs such as PuTTY. The test places can be used in manual testing that is initiated by the developer, and in feature development of the base station modules. Most of the test places that are connected to the system do not have a monitor connected to them. Thus, the remote usage is the only way to manage these test places.

Remote Desktop Connection is a remote desktop access software developed by Microsoft. The Remote Desktop connection provides the means to connect to a remote Windows PC and view the remote PC's desktop. This may be helpful in case of troubleshooting and resolving problems on a computer that is not located in the same premises [22.]. Figure 22 below is RDC's main window.

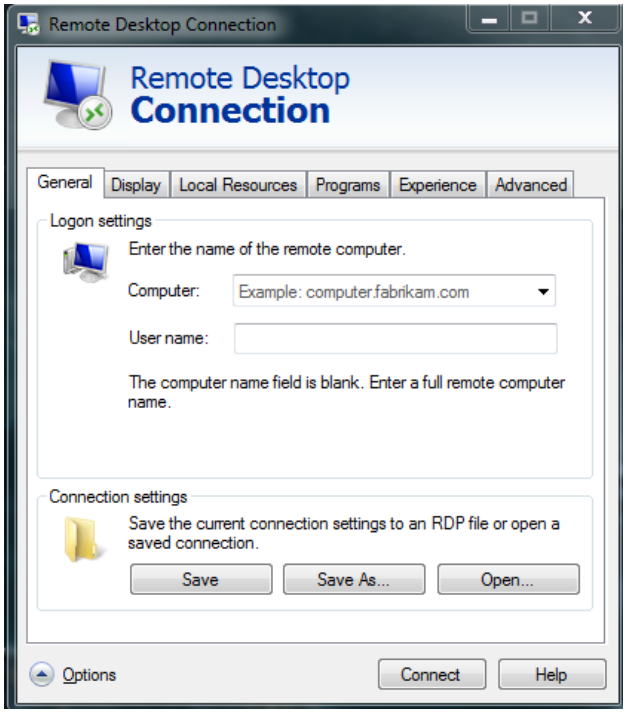

Figure 22. Remote Desktop Connection main window. Reprinted from [23.]

NoMachine NX is also a remote desktop access software and it provides similar functionality as Remote Desktop Connection. NoMachine NX is developed and maintained by a company called NoMachine. Figure 23 below is the NoMachine client's main window that shows the connections that have been previously made. NoMachine NX provides a working client for Windows, Linux and Mac OS X operating systems as well as Android and iOS. [24.]

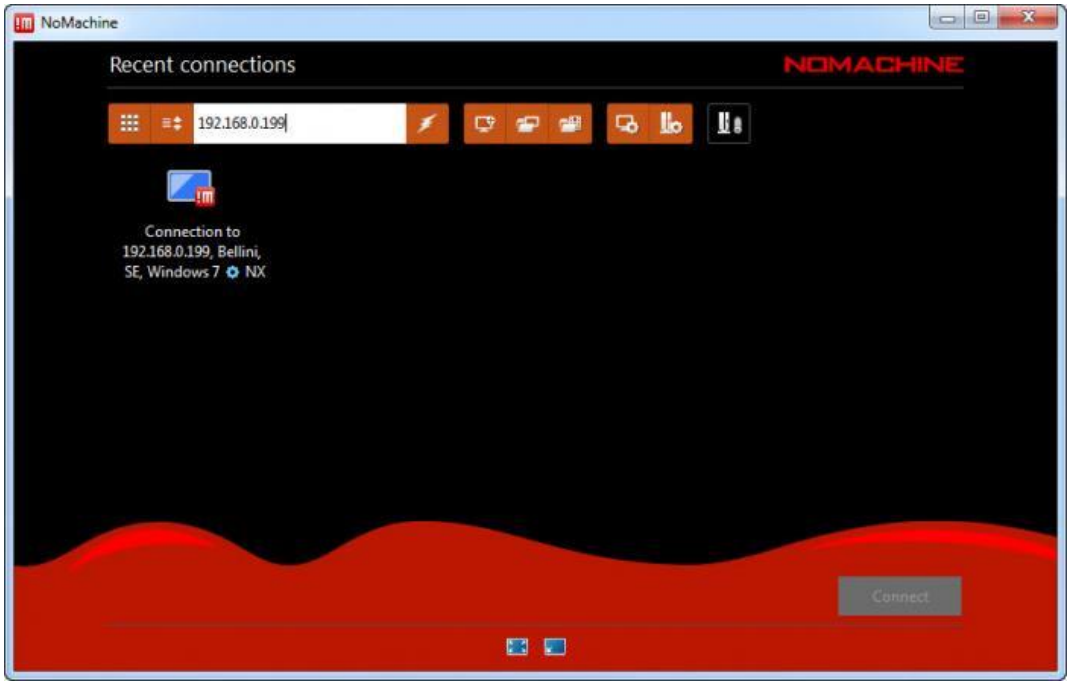

Figure 23. NoMachine main window. Reprinted from [24.]

System administrators often have the need to access and manage servers or work-stations from distant locations. Often if the system administrator manages tens or hundreds of computers, these computers may have been located in a specific room built for them called a server room. It may be a lot more effective to access those computers remotely via the system administrator's computer rather than physically relocating to the computer's location. [25.] Another purpose for the remote desktop and remote access usage is the need to perform demanding computation such as video rendering or mathematical computation.

In information technology companies the access to a company's internal network is often provided even though user is not physically located in the company's premises. This is achieved by having remote access solutions. Studies have shown that by enabling the

usage of remote access in a company, the stress of employees is reduced and companies have been able to cut down expenses by reducing office space requirements. [26.] Connecting to a company's private network is achieved by having a virtual private network (or VPN) connection over the public network. This public network is often the Internet. Figure 24 below displays the two types of remote access connections, the connection to remote access server by using dial-up and the connection by using a VPN.

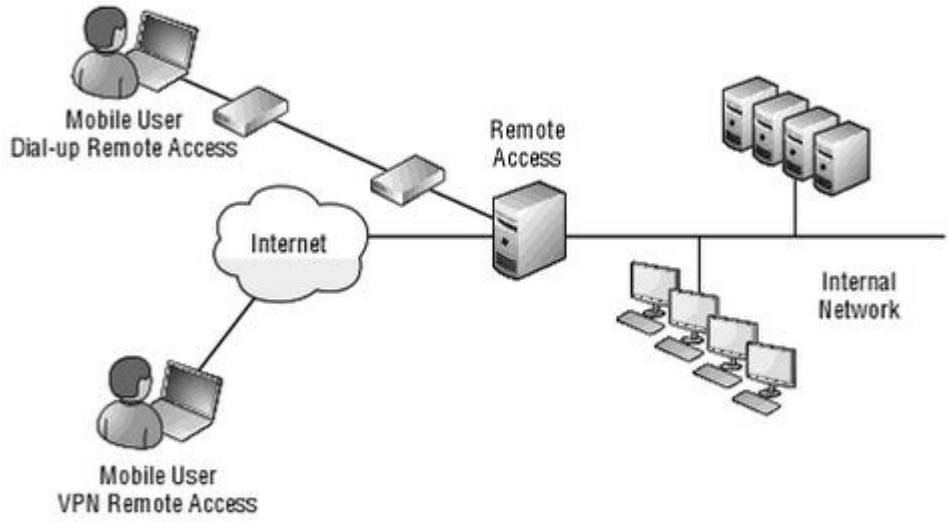

Figure 24. Remote access. Reprinted from [26.]

VPN connections use tunnelling protocols. These protocols include encryption of the connection and thereby provide additional protection for the connection. In the Windows 7 environment the following tunnelling protocols are supported: [26.]

- Internet Key Exchange version 2 (IKEv2) which is the newest tunneling protocol and was introduced with Windows 7 and Windows Server 2008. It can also provide an additional choice over PPTP by going through network address translation or NAT.
- Secure Shell Tunneling Protocol (SSTP) which was introduced with Windows Vista and Windows Server 2008. SSTP encrypts the traffic as HTTPS traffic by using SSL. SSTP provides NAT support if the VPN server is located behind NAT.
- Layer 2 Tunneling Protocol (L2TP) which was developed by combining the strengths of Microsof's PTTP with the strengths of Cisco's Layer 2 Forwarding (L2F) protocol. L2TP is supported in Windows 2000 or later version.
- Point-to-Point Tunneling Protocol (PPTP) which is the oldest of the four protocols. It encrypts data using Microsoft Point-to-Point Encryption (MPPE). PPTP is supported in Windows 2000 or later versions.

IKEv2 and SSTP both require the use of a Public Key Infrastructure to issue certificates. The Public Key infrastructure (or PKI) is a protocol that supports the distribution and identification of the public encryption keys. This allows users and computers to securely exchange data over public networks such as the Internet. Well-known public key algorithms are RSA, DSA and SHA-1. IKEv2, L2TP/IPSec and SSTP provide several important security protections such as data confidentiality by encrypting the data and ensuring the integrity of the data. [27; 28.]

4.6 The old reservation system

Previously the states of test places have been determined by different values in the database. As the database has been under changes throughout its existence it now contains obsolete data which is not used anywhere. One of the goals for the new specification was that there would be clear fields which explain the status of a particular test place.

The reservation system contains a certain set of commands that can be issued from the RemoteUser, RxUser or even from the CiMonitor. The functionality of this set of commands wanted to be kept unchanged from the user's point of view even though the auxiliary programs' logic and the internal structure of the database table were changed. Figure 25 below demonstrates how and what values were changed in the database in the old system when the user gave a specific command from RemoteUser.

Start Maintenance (+ BTS OFF)	
Column	**Value**
Node_status	ONLINE -> OFFLINE
Slave_status	ONLINE -> OFFLINE
Terminate	IDLE -> RESTORE
Pc_Desc	Add text ”(maintenance)” to the end
Reserve PC	
Column	**Value**
Current_user	Username
Lease_purpose	Purpose that the test place is used for
Lease_end_time	User defined time when the reservation ends
Lease_start_time	Current time
Release PC from CI now and **Release PC from CI until 20:00**	
Column	**Value**
Node_status	ONLINE -> OFFLINE
terminate	X -> RESTORE
Jenkins_start_time	Date and time which is this day at 8pm
Jenkins_stop_time	Date and time which is the next day at 7am
Jenkins_temp	0

Figure 25. How were the database values changed after a certain command was issued

Figure 26 below demonstrates which values were changed in the old system similarly than Figure 25 above but when the command was issued from the RxUser.

Disable CI usage	
Column	**Value**
Node_Status	ONLINE -> OFFLINE
Terminate	IDLE -> RESTORE
Jenkins_start_time	Date and time which is this day at 8pm
Jenkins_stop_time	Date and time which is the next day at 7am
Jenkins_temp	1 -> 0
Enable CI usage	
Column	**Value**
Node_status	OFFLINE -> ONLINE
Terminate	RESTORE -> IDLE
Jenkins_start_time	This date and time - 5 min. For example if it is currently 8:40 then this value will be 8:35
Jenkins_stop_time	This date and time + 1 year. For example if it is currently 14th January 2015, 8:05 then this value will be 14th January 2016, 8:05
Jenkins_temp	0 -> 1
Change CI permission	
Column	**Value**
jenkins_allowed	0 <-> 1

Figure 26. How were the database values changed after a certain command was issued

Some of the commands displayed in Figure 26 have a similar function as if they were given from RemoteUser. CIAPP also updates the database while testing is ongoing in the test place. For example, the test_status will only be set by CIAPP and this test_status is displayed by CiMonitor. The value of the test_status indicates which phase of the testing is ongoing at the moment.

4.7 The implementation of the new slave state specification

Figure 27 represents the initial idea of what the new database table should look like and what the relationships would be between the columns. Every bubble represented in this figure represents a database column. The bold text inside each of these bubbles represents the database column name. The values separated by a comma in parentheses below the name are the possible enumerated type values of that particular column.

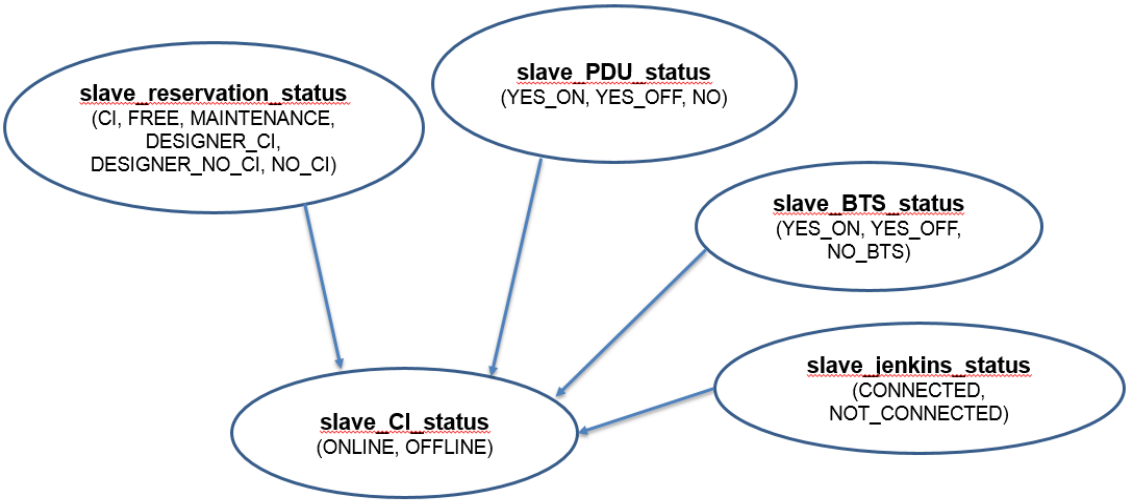

Figure 27. How the states affect each other according to the new specification. Reprinted from [14.]

The idea behind figure 27 was to have four new database columns that are slave_reservation_status, slave_PDU_status, slave_BTS_status and slave_jenkins_status which all would affect the value of slave_CI_status. This means that it would be possible to determine the state of a particular test place from the continuous integration point of view by checking only one of the database columns. Previously the continuous integration system determined whether or not to use the test place in testing by checking multiple database columns.

In order for the system to function with the new database columns, the logic of RemoteUser, RxUser, CiMonitor and Watchdog has to be changed. Every possible command that was issued from RemoteUser, RxUser or CiMonitor changed the values of one or several of the columns represented in figure 27. When, for example, a user issued a free command from RxUser in a reserved test place in the old system the values of node_status and slave_status were set to ONLINE if they were OFFLINE, indicating that the continuous integration usage was enabled in that particular test place. In order for the slave_reservation_status to indicate whether or not the user has enabled the continuous integration usage in test place the logic of the functions handling the enabling and disabling continuous integration usage has to be modified. Although the internal logic of the states was changed the functionality seen from the user's point of view needed to be unchanged. It was decided early in the project to drop out the slave_PDU_status as it was not seen as information that was reasonable to have. The initial specification was

also modified as the project progressed. Possible values for the new columns are represented in figure 28 below.

slave_CI_status	slave_reservation_status	slave_BTS_status	slave_jenkins_status
ONLINE	FREE	BTS_ONLINE	CONNECTED
OFFLINE	CI	BTS_OFFLINE	NOT_CONNECTED
	DESIGNER_CI	NO_BTS	
	DESIGNER_CI_NIGHT		
	DESIGNER_NO_CI		
	MAINTENANCE		
	NO_CI		

Figure 28. The possible values of the new columns

Each of the values shown in figure 28 is stored in the database as a MySQL type of enumerated value. Figure 29 below describes the conditions that will cause the slave_CI_status to be set in offline state in the database.

	slave_CI_status
slave_reservation_status	
MAINTENANCE	OFFLINE
DESIGNER_CI_NIGHT	OFFLINE
DESIGNER_NO_CI	OFFLINE
NO_CI	OFFLINE
slave_BTS_status	
NO_BTS	OFFLINE
slave_jenkins_status	
NOT_CONNECTED	OFFLINE

Figure 29. Conditions that cause the slave_CI_status to be set to offline in the database

This means that when the slave_reservation_status is in “MAINTENANCE”, “DESIGNER_CI_NIGHT”, “DESIGNER_NO_CI” or “NO_CI” state then the slave_CI_status will be set to “OFFLINE”. In this case the slave_CI_status will be “OFFLINE” regardless of the values of slave_BTS_status or slave_jenkins_status. The slave_CI_status differs from the other new columns in a way that the status of slave_CI_status will not take into account any of the external settings or commands. Only the combination of slave_reservation_status, slave_BTS_status, slave_jenkins_status will determine the slave_CI_status. The default values of the new database columns can be seen in figure 30 below.

Whenever a new test place is introduced to the system and Watchdog is started, Watchdog will automatically start to poll the base station status, Jenkins connection status and wait for the reservations and then update these values accordingly.

	Default value
slave_CI_status	OFFLINE
slave_reservation_status	FREE
slave_BTS_status	NO_BTS
slave_jenkins_status	NOT_CONNECTED

Figure 30. The default values of the new database columns

The connection to Jenkins is handled by having a Java application that establishes the connection. This application uses port 3141 for communication. This number of this port is defined in Jenkins' system configuration. Since the new system should detect whether or not the test place is connected to Jenkins and update that value to the database the status of port 3141 has to be read. This can be achieved rather easily by using a Python module called psutil, which is a cross-platform library for fetching information about current processes run on the system. The psutil utilizes many command line tools provided by the operating system such as ps, top, lsof, netstat and ifconfig. Below, in listing 2, there is a code example of how to test the usage of port 3141. [29.]

```
connections = psutil.net_connections()
for conn in connections:
    if conn[3][1] == 3141:
        jenkinsFound = True
        break
```

Listing 2. A code example illustrating how to check the Jenkins connection status

The function call `net_connections()` that is provided by the psutil module returns an array that contains extensive information about every socket connection that is present in the system. The array contains information such as:

- the file descriptor of a particular socket
- the address family of the socket
- the address type of the socket

- local and remote address
- status of the socket as a constant value such as `CONN_CLOSE` or `CONN_LISTEN`
- process id of the socket

The array value `conn[3][1]` that is tested in listing 2 is the local address port number while `conn[3][0]` would have been the local IP address. The array also contains the status of the current socket which in this case is `psutil.CONN_LISTEN` that indicates that the connection has been established and working. If, however, the test place PC drops out of network when for example a network cable is removed the status of the socket stays in `psutil.CONN_LISTEN`. This means that by checking only the status value in order to determine the connection status with the Jenkins server is not entirely reliable. However, if the physical connection to the network is not present Watchdog is not able to update the status to the database so the current solution can be considered to be fit for its current usage. The psutil library defines many constants for describing the connection. When the Java application that holds the connection with Jenkins is closed port 3141 is also no longer used. [26.]

4.8 Results

As a result of the work done for this project a new test place reservation system was developed. The functionality of the new modifications were tested using a testing environment that has been established for testing primarily the modifications of the CIAPP. This testing environment allowed the simulated continuous integration process to be run in order to be confident that the modifications were working, and that there were no clear logic flaws in the code. These test runs consisted of an actual continuous integration process to be run with an actual test case to be executed. This test case that was run only printed out environmental information and created a dummy file in order to simulate the actual testing process. By running the simulated continuous integration process the system used the new logic to determine which of the test places were available for testing, and on which test place the tests were then run on.

During this project the new slave state specification has undergone minor changes. These changes were introduced in order to have only relevant information stored in the database from the continuous integration point of view. Also for example the calculation

of the continuous integration usage during the user reservation was re-implemented. The new specification would have not affected the implementation of the calculation, but it was decided to be changed since this project allowed the transition to be smooth.

5 Conclusion

The goal of this project was to modify the logic of the in-house programs that were used in test place management. These test places are used in continuous integration to test the development code. The modifications were made according to a pre-defined slave state specification. The idea behind the new specification was to simplify the overall complexity of the continuous integration and the in-house programs' logic by introducing five objects that hold information about each test place.

As a result, a new test place reservation system was developed. The new specification has undergone minor changes during this project. These modifications to the specification were made in order to have only relevant information stored in the database. Continuous integration availability can now be determined by only checking a single value. Also from now on determining the latest status of a test place from the database in case of errors is more convenient. Since the implementation to a production pipeline would require the continuous integration process to be stopped, this new system was not taken into use during the writing process of this thesis.

As a future development this system could be done entirely as a real time system using only network sockets without any kind of dependency with an external database. This kind of real time approach would also contain some pitfalls as will this current system. Real time approach would decrease the delay that may have been caused by the high usage of the database.

References

1 Cox, Christopher. Introduction to LTEs: LTE, LTE-Advanced, SAE and 4G Mobile Communications (2nd Edition). Chichester, United Kingdom: John Wiley & Sons; 2012.

2 Brydon, Alastair. Video Streaming and Downloads to Dominate Mobile Service Mix. [online]. August 2010
URL: http://www.unwiredinsight.com/2010/video-streaming-3g-mobile. Accessed 30 March 2015.

3 Fowler, Martin. Continuous Integration [online]. May 2006.
URL: http://www.martinfowler.com/articles/continuousIntegration.html. Accessed 5 January 2015.

4 Demarey, Christophe. What is Continuous Integration? [online].
URL: http://chercheurs.lille.inria.fr/~demarey/Main/ContinuousIntegration. Accessed 22 April 2015.

5 Cooke, Jamie Lynn. Everything You Want to Know About Agile: How to Get Agile Results in a Less-than-Agile Organization. Ely, United Kingdom: IT Governance; 2012.

6 Waterfall Development Methodology [online].
URL: http://learnaccessvba.com/application_development/waterfall_method.htm. Accessed 22 April 2015

7 Arunraj, R. A Short Intro to Scrum Methodology [online]. June 2014.
URL: https://www.linkedin.com/pulse/20140612164300-225306444-a-short-intro-to-scrum-methodology. Accessed 22 April 2015

8 Acharya, Sujoy. Test-Driven Development with Mockito. Birmingham, United Kingdom: Packt Publishing Ltd; 2013.

9 PuTTY [computer program]. Version 0.64. Cambridge, United Kingdom: Simon Tatham; 1997.

10 Kawaguchi, Kohsuke. What is Jenkins? [online]. November 2013.
URL: https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins. Accessed 12 January 2015.

11 Cron. [online]. December 1999.
URL: http://www.unixgeeks.org/security/newbie/unix/cron-1.html. Accessed 16 March 2015.

12 Harila, Mikko. Team leader. Nokia Oyj, Espoo. Conversation. 24 March 2015

13 Firefox [computer program]. Version 31.6.0. Mountain View, California, United States of America: Mozilla Foundation: 2002.

14 Nokia Oyj. Confidential in-house documentation. Espoo, Finland: Nokia Oyj; 2012

15 Harwani, B.M. Introduction to Python Programming and Developing GUI Applications with PyQT. Boston, MA, United States of America: Course Technology / Cengage Learning; 2011.

16 RemoteUser [computer program]. Espoo, Finland: Nokia Oyj; 2012.

17 RxUser [computer program]. Version 3.61.0.0. Espoo, Finland: Nokia Oyj; 2012.

18 CiMonitor [computer program]. Espoo, Finland: Nokia Oyj; 2012.

19 MySQL. The TIME Type. [online].
URL: http://dev.mysql.com/doc/refman/5.0/en/time.html. Accessed. 10 March 2015.

20 MySQL. Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT. [online].
URL: http://dev.mysql.com/doc/refman/5.1/en/integer-types.html. Accessed 10 March 2015.

21 Microsoft Office Outlook 2007 [computer program]. Microsoft; 2007.

22 Mueller, John Paul. Windows Command Line Administration Instant Reference. Canada: Sybex; 2010.

23 Fullmer, Steve. Enabling Windows 7 Remote Desktop and Remote Assistance. [online]. September 2012.
URL: http://blogs.interfacett.com/enabling-windows-7-remote-desktop-remote-assistance. Accessed 22 April 2015

24 NoMachine. [online]
URL: https://www.nomachine.com. Accessed 20 February 2015

25 IBM Redbooks. Implementing IBM Tivoli Remote Control Across Firewalls. United States of America: IBM; 2003

26 Gibson, Darril. Windows 7 Desktop Support and Administration: Real World Skills for MCITP Certification and Beyond (Exams 70-685 and 70-686). Chichester, United Kingdom: John Wiley & Sons; 2010.

27 Rouse, Margaret. PKI (Public Key Infrastructure). November 2014.
URL: http://searchsecurity.techtarget.com/definition/PKI. Accessed 23 January 2015.

28 Adams, Carlisle, Lloyd, Steve. Understanding PKI: Concepts, Standards, and Deployment Considerations. Boston MA, United States of America: Addison-Wesley Professional; 2003.

29 Psutil 2.2.0 Documentation [online].
URL: http://pythonhosted.org/psutil/. Accessed 14 February 2015